2nd Hellenic Conf. on AI, SETN-2002, 11-12 April 2002, Thessaloniki, Greece,
Proceedings, Companion Volume, pp. 475-486

SpiderServer : the MetaSearch Engine of WebNaut

Nick Zacharis and Themis Panayiotopoulos

Knowledge engineering Lab, Department of Informatics,
University of Piraeus, Piraeus, 185 34, Greece
nzach@unipi.gr, themisp@unipi.gr,

Abstract. Search engines on the Web are valuable tools for searching information according to a user's interests whether an individual or a software agent. In the present article we describe the design and the operation mode of SpiderServer, a metasearch engine used for the submission of a query followed by the retrieving of results from five popular search engines. SpiderServer is the metasearch engine of the WebNaut system but it can be easily used by any other metasearch platform. There are two files for every search engine describing the phases of query formation and filtering respectively. These files contain directions on the way a query must be modified for a specific search engine and on the methodology SpiderServer must follow in order to parse the results from the specific search engine. The ultimate goal is to construct platform independent meta-search engines, which can be easily programmed to adapt to any search engine available on the WEB.

1 Introduction

Most users do not use the large quantity of information available on the web unless it is indexed in one or most of the search engines available on the web. Through an interface, users submit queries - usually a combination of keywords and logic operators - to the search engine, which in turn, collects from the database all the urls related to the specific query.

Web search technologies have been recently classified, [1,2], into six basic categories : hyperlink exploration, information retrieval, metasearches, SQL approaches, content-based multimedia searches, Artificial Intelligence based searches, etc. There are many well known Web search systems, such as Altavista, Excite, Hotbot, Lycos,Yahoo, [3-7] etc., each one of which uses various techniques, for searching the web, [1,2].

However, it seems that none of the current search engines is capable of providing a thorough Web coverage with full up-to-date Web information. Metasearch engines, [8,9,10], have been developed to overcome this difficulty. Such engines conduct a search by posting a query to other search engines, and receiving the best results, which then present to the user.

The interfaces of the currently available search engines have not followed the research towards the direction of standardization of internet retrievals and returns, [11], and therefore the development of metasearch engines becomes a very difficult task.

During the last two years we have developed a personalized software agent for the retrieval of information available on the web in accordance with the user's profile, [12,13]. The Webnaut system learns the user's interests and adapts appropriately as these interests change over time. The learning process is driven by a metagenetic algorithm along with the user feedback to an intelligent agent's filtered selections.

2. The WebNaut System

Users provide examples of web pages which most appropriately describe their interests and the WebNaut's agents, after developing the user's profile based on these examples, tries to find other pages matching it.

WebNaut consists of a set of interconnected agents : ProxyServer agents, SiteMirror agent, WWWserver agent, Metasearch agent, and a Learning agent. Each agent has a specific job and contributes through a common interface to produce a modular but integrated system, [12].

User profiling is performed by the Learning Agent of the WebNaut system, [13]. All the words included in a document collection are extracted to create a dictionary vector. A weight is related to each word, indicating the number of documents that contain the keyword. Moreover, the sum of keyword frequencies in all the texts it appears is also computed. In this way, WebNaut maintains a Nx3 matrix Dictionary with keywords, weights and frequencies, and uses this Dictionary as the user's profile. A two level genetic algorithm creates complex queries using words from the dictionary and logical operators, which are send to the Metasearch engine.

The SpiderServer MetaSearch engine posts these queries to well known search engines, takes their results and stores them in a response-list. Every new result is compared to previous ones and if it has been already stored it is not inserted in the list. In this way, the results from many search engines are gathered by WebNaut.

A similarity function is used to evaluate the similarity of a document to the profile. This means that no classification takes place, as WebNaut is more of a personal intelligent assistant and not a document classifier. Documents are evaluated towards the interests of the user, and not towards a concept hierarchy scheme.

Finally, the most fitting documents and presented to the user. The user evaluates them further, and the learning agent updates the users' profile. In this way, the learning agent can create complex representative structures of the users' long term interests.

Webnaut was initially developed by using a single search engine for the information collection task, and the whole process of forming a query, requesting the relevant information and parsing the results of the search engine, were encoded inside the program.

When in time the program ceased to offer results – because the search engine's administrator changed the HTML form, as well as the format of the results- it became clear that we had to choose another way of coding. At the same time, the following questions came to the fore:

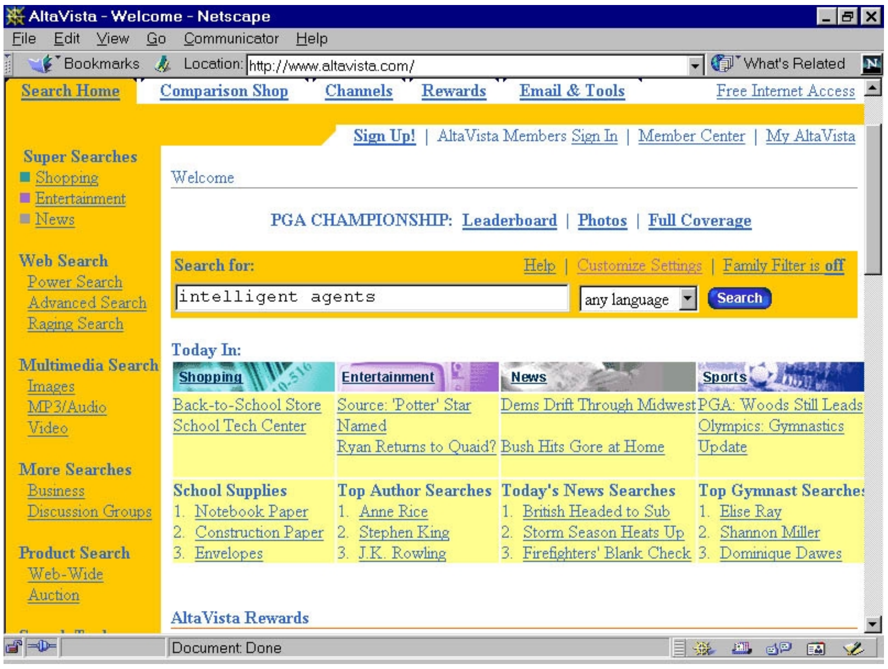
a) is a single search engine able to cover all areas? Should we use more than one search engines?

b) how can a software agent draw information from a search engine or a database for which, apart from its url, it does not have any other information?

All the above problems and questions motivated us to become more involved in the issue and develop a meta-search engine, the SpiderServer. During the design process we discovered that the specific combination of describing the features of a database, as well as the way of collecting information from the results produced by the database, could be used in other domains related to the collection of information from the web; for instance, in electronic auctions, on-line shopping etc. In the present article we will describe the design and operation of this meta-search engine and will of course discuss its advantages and disadvantages.

3. Transaction between a web-browser and a search engine

The transaction between a user's web-browser and the Web Server of a search engine is based on the HTTP, [3], request/response model. The client sends its request to the server and waits for the server's response. Generally, during the communication of a client-server most requests refer to static HTML files in the server's hard disk. However, in many web applications as well as in the case of a search engine, there are no static files but dynamic web pages generated on the fly. The submission of a request to the server is accomplished through an HTML form, [4], using one of the two methods of submission, GET or POST.



Location: http://www.altavista.com/

Search for: intelligent agents

any language

Fig. 1. The search interface of AltaVista search engine.

Figure 1 shows the interface of the popular search engine AltaVista, [5]. Users type the keywords for the subject of their interest i.e. *intelligent agents*, in the *Search for* text box. Then by pressing *Search* button the browser carries out the submission of the HTML form to the server and more specifically to the URL indicated by the ACTION parameter within the <FORM> statement. A few lines of HTML code used by AltaVista for the collection of the user's input is shown in Figure 2. As we can see in the code, Alta Vista's engine provides the opportunity to search and translate in a specific language. The options offered by search engines vary, but this issue will be discussed later in this article.

To simplify but without loosing sight of the issue, we will consider that the server undertakes all the above operations whereas in reality the server executes a CGI script which in its turn, after receiving the request's parameters, follows the necessary processing. AltaVista's server, after working through the request, responds with a list of results as the ones shown in figure 3.

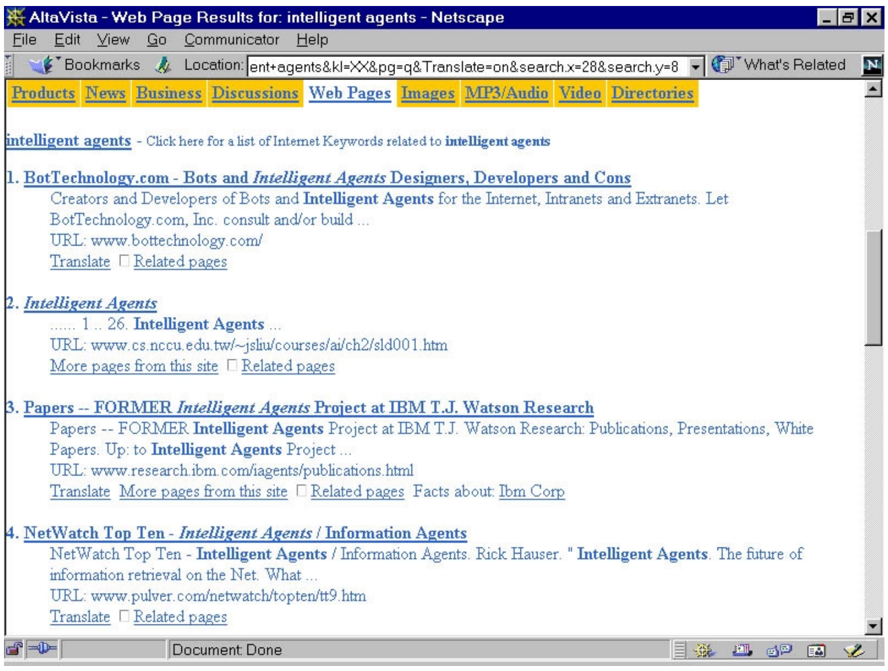
In receiving the request, the server will do the following tasks: a) parsing the user's input b) process the request; this stage can be fairly complicated, including communication with other servers, databases etc. c) formatting and sending the results to the client.

```
<form action="/cgi-bin/query" name=mfrm>
<input type=text name=q size=35 maxlength=800 value="">
<SELECT NAME=kl>
<OPTION VALUE=XX SELECTED>any language
...............................................
<OPTION VALUE=el>Greek
<OPTION VALUE=he>Hebrew
...............................................
</SELECT>
<input type=image name=search src= src=http://a12.g.akamai.net/7/search.gif" alt="Search">
<INPUT TYPE=hidden NAME=pg VALUE=q>
<INPUT TYPE=hidden NAME=Translate VALUE=on>
</form>
```

Fig. 2. The HTML source that was used to generate the AltaVista's query form

For every result we use the term *result record* wishing in this way to define a set of information made up of the following elements: an url, a description title and a summary. More specifically, in figure 3 the first result record is defined as follows:

Url	:	http://www.bottechnology.com/
Description	:	BotTechnology.com - Bots and Intelligent Agents Designers, Developers and Cons
Summary	:	Creators and Developers of Bots and Intelligent Agents for the Internet, Intranets and Extranets. Let BotTechnology.com, Inc. consult and/or build,


AltaVista - Web Page Results for: intelligent agents - Netscape
File Edit View Go Communicator Help
Bookmarks Location: ent+agents&kl=XX&pg=q&Translate=on&search.x=28&search.y=8 What's Related
Products News Business Discussions Web Pages Images MP3/Audio Video Directories
Document: Done

intelligent agents - Click here for a list of Internet Keywords related to **intelligent agents**

1. BotTechnology.com - Bots and *Intelligent Agents* Designers, Developers and Cons
Creators and Developers of Bots and **Intelligent Agents** for the Internet, Intranets and Extranets. Let BotTechnology.com, Inc. consult and/or build ...
URL: www.bottechnology.com/
Translate Related pages

2. *Intelligent Agents*
...... 1 .. 26. **Intelligent Agents** ...
URL: www.cs.nccu.edu.tw/~jsliu/courses/ai/ch2/sld001.htm
More pages from this site Related pages

3. Papers -- FORMER *Intelligent Agents* Project at IBM T.J. Watson Research
Papers -- FORMER **Intelligent Agents** Project at IBM T.J. Watson Research: Publications, Presentations, White Papers. Up: to **Intelligent Agents** Project ...
URL: www.research.ibm.com/iagents/publications.html
Translate More pages from this site Related pages Facts about: Ibm Corp

4. NetWatch Top Ten - *Intelligent Agents* / Information Agents
NetWatch Top Ten - **Intelligent Agents** / Information Agents. Rick Hauser. " **Intelligent Agents**. The future of information retrieval on the Net. What ...
URL: www.pulver.com/netwatch/topten/tt9.htm
Translate Related pages

Fig. 3. Sample response of AltaVista for the query '*intelligent agents*'

```
.............
<b class=txt2>1.</b>
<b>
<a href="http://www.bottechnology.com/">BotTechnology.com - Bots and <EM>Intelligent
Agents</EM> Designers, Developers and Cons</a></b>
<dd>
Creators and Developers of Bots and <B>Intelligent Agents</B> for the Internet, Intranets and
Extranets. Let BotTechnology.com, Inc. consult and/or build ...<br>
<span class=ft>
 URL: www.bottechnology.com/
</span>
.............
<b class=txt2>2.</b>
<b>
<a href="http://www.cs.nccu.edu.tw/~jsliu/courses/ai/ch2/sld001.htm"><EM>Intelligent
Agents</EM></a></b>
<dd>
...... 1 .. 26. <B>Intelligent Agents</B> ...<br>
<span class=ft>
 URL: www.cs.nccu.edu.tw/~jsliu/courses/ai/ch2/sld001.htm
</span>
.............
```

Fig. 4. The HTML source of the AltaVista's response for the query 'intelligent agents'.

Figure 4 illustrates only the code referring to the result records 1 and 2. If an application was going to extract and use only the information appearing in the result records, it would have to ignore all the html tags and irrelevant information (i.e. advertisements, help icons etc.) during the parsing stage and keep only the information referring to the fields of result record.

Meta-search engines belong to this type of applications that do not maintain any local database but operate entirely by using the index of other search engines through a common interface. Meta-search engines attempt to solve the problems of a single search service, such as the outdated index, limited coverage etc.

The user submits a query to the meta-search engine and that in its turn promotes the query to the search engines; either to one after the other, or to all at the same time. Results from each search engine are collected by the meta-search engine and merged in one file either according to their list appearance (e.g. the first result from each search engine then the second one and so-on) or all the results per search engine.

4. Implementation

4.1 The SpiderServer Architecture

From the previous section it became clear that search engines differentiate themselves in both, the level of encoding HTML form, meaning that every search engine uses a different method to collect user's input, different FORM tags etc., and in relation to the choices offered i.e. the opportunity to translate, to use boolean expressions etc. In addition, search engines differentiate themselves in another level, that of result page presentation requiring the use of a specific script for the parsing of results for each search engine. Thus, we could divide the implementation of a meta-search engine in two phases as they appear in Figure 5.

Phase 1: We could name this the query phase, as during its operation the user's question will be formed according to the standards of every search engine. In addition, during this phase the submission of the query to the specific search engine will be accomplished. The first stage of the first phase will be the analysis of the user's query. During this stage, all data from the user's request are collected and in case of invalid or missing data, the user is redirected to an error page. The applicable error message informs the user the reason for the disclaimer of the request's fulfillment.

During the *Query Construction* stage, the user's query is transformed to a query with logic operators, which is acceptable by the search engine used. For instance, some search engines accept the word AND for the conjunction of keywords whereas others accept the symbol +. In our implementation, we support the following operators "AND", "OR", "NOT", "EXACT", "EXACT NOT", when of course they are supported by the respective search engine.

The last stage of the first stage is the *Query Submission* stage where the metasearch engine enhances the query with specific FORM tags and by using the indicated

FORM method, forwards the query to the script indicated by the ACTION parameter within the <FORM> statement.

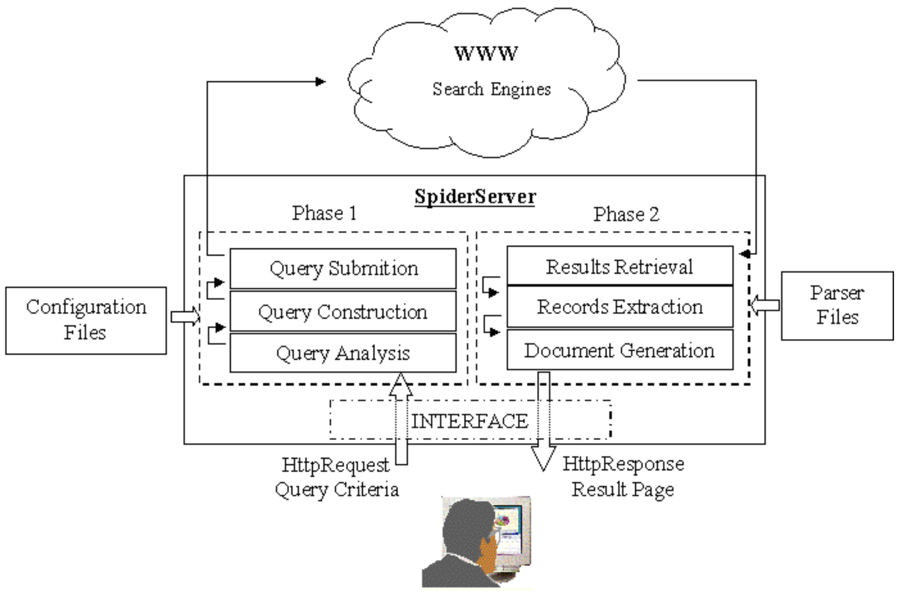


Fig. 5. The architecture of SpiderServer

Phase 2: This phase follows the previous one and we could name it parser phase. During the second phase we will collect the results from every search engine. Since every search engine uses a different way of result presentation, a different script must be used for the extraction of result records.

Similar to the first phase, this one is also divided in three stages. The *Result Retrieval* stage involves fetching the result pages of each search engine for processing. During the next stage the result records are parsed from the result pages. Finally, the *Document Generation* stage, involves organizing and presenting the results that each search engine provides in one of the available formats. In current implementation the default presentation format is <title><url><description>. Also, during this stage the results that have the same title or/and URL are removed, if of course the user wishes it.

Figure 5 also shows the whole operation where the user addresses queries through an interface. Aiming to avoid direct access to the implementation code for the support of future changes by the search engine administrator, we thought of using two template files for the description of every search engine, *Option.txt* - which describes general elements for every search engine as well as its available options - and *Parser.txt* – which describes the way we will achieve parsing the result records from the specific search engine. These two files will be used by the metasearch engine in order to form and submit the user's or the agent's query, to the specific search engine and then, for the analysis of its results.

In the following paragraphs we will discuss the interface of the implementation as well as the template files and their use. The interface of this specific application is a rather important factor for the success or not of the whole implementation as it should not restrain the user and should be as flexible as possible so as to allow the incorporation of new services provided by the search engines. Furthermore, the presentation of interface is valuable for two more reasons: a) It allows the user of the implementation to describe the query in a general format which will be then adapted to the requirements of every search engine and b) it allows the presentation of the code used in the various stages of the implementation to become more explicable.

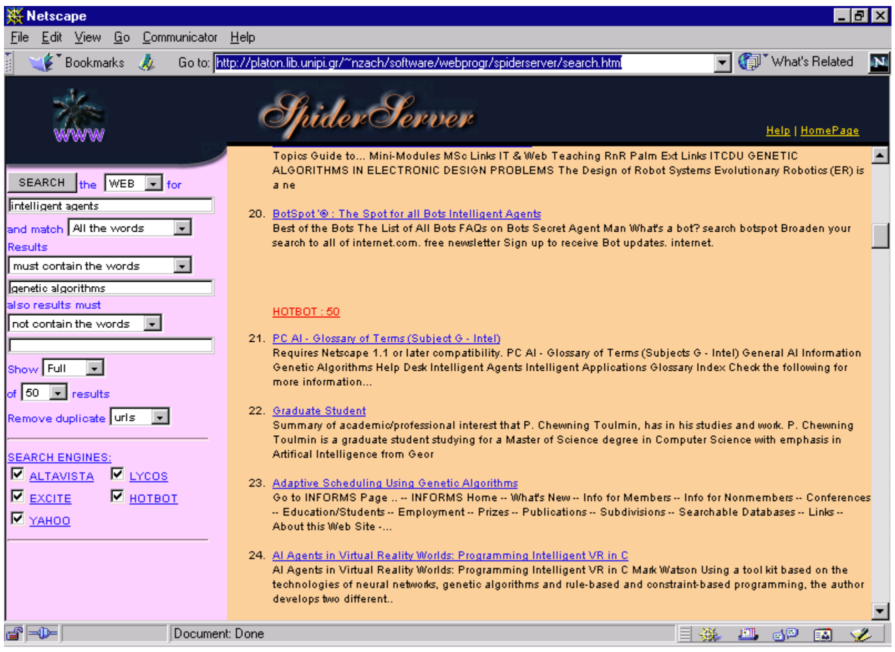
4.2. The SpiderServer Interface

Current implementation offers a common query interface to five popular web search services, AltaVista [5], Excite [6], Hotbot [7], Lycos [8], Yahoo [9]. The query interface supports basic logic operators such as AND, OR, NOT as well as advanced operators such as 'EXACT' and 'EXACT NOT' phrase. Figure 6 shows the public query interface for the SpiderServer, where the users can choose which search service to run, how many hits to retrieve, the appearance of the results and so on.

The interface of SpiderServer is based on HTML frames and forms, as shown in Figure 6. Frames allow us to split the browser view into multiple windows and to display a separate html document in each window. Moreover, actions and scripts in one frame can be programmed to control and update the content of adjacent frames. Frames are popular with Web page designers because of their properties, offering new possibilities in information presentation as well as site navigation.
The following piece of HTML code implements the desirable frame based interface on client's browser and allows queries to be submitted and answered on the same page, with one frame called "QUERY" holding the query form, and the other called "RESULTS" presenting the results of the query form, when it is submitted to the SpiderServer.

```
<html>
<frameset rows="10%,*" border=0>
        <frame src="Logo.html" name="LOGO" SCROLLING="no" NORESIZE>
        <frameset cols="33%,*" border=0>
        <frame src="Query.html" name="QUERY" SCROLLING="auto" NORESIZE>
        <frame src="Results.html" name="RESULTS" SCROLLING="auto" NORESIZE>
        </frameset>
</frameset>
</html>
```

The third frame called "LOGO" is a constant size region and contains elements that the user must always see, such as application logos, links to help pages, etc. This static frame is placed on the top of the page above of the other two frames. In the query form - see figure 6 -, the user has selected items so as to formulate a query, in order to find web documents that must contain the phrase "intelligent agents" and also these documents must contain the words "information" and "filtering" but not the words "genetic" and "algorithms".


Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://platon.lib.unipi.gr/~nzach/software/webprogr/spiderserver/search.html
What's Related
WWW
SpiderServer
Help | HomePage

SEARCH the WEB for
intelligent agents
and match All the words
Results
must contain the words
genetic algorithms
also results must
not contain the words

Show Full
of 50 results
Remove duplicate urls

SEARCH ENGINES:
✓ ALTAVISTA ✓ LYCOS
✓ EXCITE ✓ HOTBOT
✓ YAHOO

Topics Guide to... Mini-Modules MSc Links IT & Web Teaching RnR Palm Ext Links ITCDU GENETIC ALGORITHMS IN ELECTRONIC DESIGN PROBLEMS The Design of Robot Systems Evolutionary Robotics (ER) is a ne

20. BotSpot '® : The Spot for all Bots Intelligent Agents
Best of the Bots The List of All Bots FAQs on Bots Secret Agent Man What's a bot? search botspot Broaden your search to all of internet.com. free newsletter Sign up to receive Bot updates. internet.

HOTBOT : 50

21. PC AI - Glossary of Terms (Subject G - Intel)
Requires Netscape 1.1 or later compatibility. PC AI - Glossary of Terms (Subjects G - Intel) General AI Information Genetic Algorithms Help Desk Intelligent Agents Intelligent Applications Glossary Index Check the following for more information...

22. Graduate Student
Summary of academic/professional interest that P. Chewning Toulmin, has in his studies and work. P. Chewning Toulmin is a graduate student studying for a Master of Science degree in Computer Science with emphasis in Artifical Intelligence from Geor

23. Adaptive Scheduling Using Genetic Algorithms
Go to INFORMS Page .. -- INFORMS Home -- What's New -- Info for Members -- Info for Nonmembers -- Conferences -- Education/Students -- Employment -- Prizes -- Publications -- Subdivisions -- Searchable Databases -- Links -- About this Web Site -...

24. AI Agents in Virtual Reality Worlds: Programming Intelligent VR in C
AI Agents in Virtual Reality Worlds: Programming Intelligent VR in C Mark Watson Using a tool kit based on the technologies of neural networks, genetic algorithms and rule-based and constraint-based programming, the author develops two different..

Document: Done

Fig. 6. The interface of SpiderServer

Query: "intelligent agents" AND (information, filtering) NOT (genetic, algorithms)

This query is sent as a POST request by the web browser to the SpiderServer, which will respond with results that will be displayed in the RESULTS window.

Figure 6 show that the user of the application has in his/her disposal 3 textboxes in order to describe his/her query. The processing of a request by the server will begin with the reading of number of all the text boxes within the form and is described in the hidden input tag with the name *noQuery*. Then, an iteration will work through all textboxes, as the name of each of them is a combination of the word QUERY and the corresponding iteration number, e.g <INPUT NAME="QUERY1" SIZE="22" VALUE="">. In the same iteration, we will also interpret the logic operator connecting the words in the respective text field. The logical operator is described in a SELECT box whose name is a combination of the word MATCH and the corresponding iteration number, e.g. <SELECT NAME="MATCH1">.

4.3. The Configuration file

All available options of a search engine are described in a text file named *Options.txt*. Figure 7 shows the configuration file for the search engine AltaVista.

In this initial implementation, the configuration file includes the information of a search engine related to its name (WEBNAME), URL (WEBHOST), the request method (WEBMETHOD), the script (WEBLINK), the tag for the query keywords

(WEBQUERY), other tags (WEBTAGS), the boolean expressions (WEBAND, WEBOR, WEBNOT) as well as the way to submit an exact phrase query (WEBEXACT), and finally the number of results that will be returned by the search engine (WEB10, WEB25, WEB50). For every search engine known to the server**,** exists a respective configuration file. Most search engines on the Web support the above options.

TAG	**CODE**
WEBNAME	ALTAVISTA
WEBHOST	www.altavista.com
WEBLINK	/cgi-bin/query
WEBPORT	80
WEBMETHOD	GET
WEBQUERY	q
WEBTAGS	pg=aq&kl=XX&d0=&d1=&search=Search&r=
WEBAND	AND
WEBOR	OR
WEBNOT	AND NOT
WEBEXACT	"QUERY"
WEBNOTEXACT	AND NOT "QUERY"
WEB10	nbq=10
WEB25	nbq=30
WEB50	nbq=50

Fig. 7. The configuration file for the AltaVista

Now after the encoding of request to the search engine we then submit the data according to the method of data submission (GET or POST). When the submission of data is completed, we can then read the complete response by the search engine through an iteration. Finally, since we have the search engine's complete response in *Text* we can proceed with the parsing of the result records.

4.4. The parser file

From the moment the search engine returns the results we will have to analyze them as easier and faster as possible. In addition, a visual description of a result record would be desirable. It is evident from Figure 4 that the script of Alta Vista uses an iteration for the formation of the results, something, which is true for most, if not all search engines. Thus, we will have to find a way of describing a result record - something like a pattern - and then to repeat the process of matching the pattern to the result page, which is saved in *Text*.

In addition, by using the following tags we will be able to parse all the result records.

- **str (start)**: Moves indicator to string (or character) that describes the code field. Basically, we place the indicator at the beginning of result record. If there is no code string, this means that the process of result parsing is completed.
- **mch (match)**: the following characters will have to be a number, which will be followed by the string (or character) describing the corresponding code field. Basically, it is used as a guard statement for the search engines which use numbering during the presentation of their results.
- **Ign (ignore)**: Move indicator to string (or character) describing the code field.
- **Lnk (link)**: The url of result record is all the characters from the point of the indicator to that of the string (or character) describing the corresponding code field. Move indicator to code field.
- **Dsc (description)**: The description title of the result record is all the characters from the point of the indicator to that of the string (or character) describing the corresponding code field. Move indicator to code field.
- **Sum (summary)**: The summary of the result record is all the characters from the point of the indicator to that of the string (or character) describing the corresponding code field. Move indicator to code field.
- **End (end)**: Match the string (or character) describing the corresponding code field. In addition, it is used to establish that the procedure for the extraction of the fields of a result record is completed and that the parser must continue with the str tag.
- **Brk (break)**: It is used to establish that the procedure for the extraction of the fields of a result record is completed and that the parser will have to proceed with the next result record, that is, from the str tag. Basically, it is used as an exit and continue statement because there is no code field for this tag.

TAG	CODE	TYPE
str	[txt2>]	s
mch	[.]	ec
ign	[="]	s
lnk	[">]	s
dsc	[</a>]	s
ign	[<dd>]	s
sum	[<br]	s
brk	[]	

Fig. 8. The parser file for the AltaVista's result page

When the process is completed the whole operation is repeated until the indicator reaches the end of the result page or when it cannot match the corresponding code string of the str tag.

5. Conclusion

Our primary goal was to develop a metasearch engine flexible enough to be used by an ordinary user without special knowledge in programming. After studying the operation and presentation of results offered by existing search engines we developed SpiderServer. What we believe to be of considerable interest is that the entire operation of the implementation is controlled through two template files.

The advantages of an approach as such would be numerous, as users would always use the same interface –that offered by the browser- to address their queries, as they would not any longer need to learn any other combination of keywords with logic operators or learn how to compose complicated commands in order to form their queries. After that, by enquiring the two template files, they will be in a position to address queries and withdraw information from anywhere on the web.

Compared to other metasearch engines, for which the developers have published some research work, there is no similar approach, to our knowledge, that parameterises in such a way the interface to various search engines. In fact, the SpiderServer approach can be easily integrated to Agent Communication Languages, to serve as interfaces to various web search engine. In this way, when a software agent is searching for some information on the web from a specific search engine, it can request for the options.txt and parser.txt, and after retrieving them it can easily use them to access the desired information.

REFERENCES

[1] Wen-Chen Hu, "An overview of the World Wide Web search technologies," In the proceedings of 5 th World Multi-conference on System, Cybernetics and Informatics, SCI2001, Orlando, Florida, July 22-25, 2001.

[2] Wen-Chen Hu, 'World Wide Web Search Technologies', chapter of the book, Shi Nansi (Ed.), 'Architectural Issues of Web-Enabled Eloctronic Business', Idea Group Publishing,

[3] Altavista: http://www.altavista.com

[4] Excite: http://www.excite.com

[5] Hotbot: http://www.hotbot.com

[6] Lycos: http://www.lycos.com

[7] Yahoo: http://www.yahoo.com

[8] D. Dreilinger, A.E. Howe, "Experiences with selecting search engines using metasearch", ACM Transactions on Information Systems, 15(3):195-222, July 1997.

[9] D. Dreilinger, A.E. Howe, "A meta-search engine that learns which search engines to query ", AI Magazine, 18(2), 1997.

[10] E. Selberg, O. Etzioni, "The MetaCrawler architecture for resource aggregation on the Web ", IEEE Expert, 12(1):8-14, January/February 1997.

[11] L. Gravano, K. Chang, H. Garcia-Molina, C. Lagoze, A. Paepcke, "STARTS : Stanford protocol proposal for internet retrieval and search", Proceedings of the ACM SIGMOD International Conference on Management of Data, 1997.

[12] N. Z. Zacharis and T. Panayiotopoulos, 'Web Search Using a Genetic Algorithm', IEEE Internet Computing, 5(2), 18-26, (2001)

[13] N.Z. Zacharis and T. Panayiotopoulos, "A metagenetic algorithm for information filtering and collection from the World Wide Web", Expert systems – The International Journal of Knowledge Engineering, Vol. 18, No 2, pp.99-108, May 2001.

[14] HyperText Transfer Protocol (ver 1.0) : http://www.ietf.org/rfc/rfc2616.txt

[15] HTML FORMS: http://www.w3.org/TR/html4/interact/forms.html